An Automation Language for Managing Operations in the Deep Space Network (ALMO)[1]

Patricia F. Santos, Paul Pechkam

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 156-201
Pasadena, CA 91109-8099
{Patricia.F.Santos, Paul.Pechkam}@jpl.nasa.gov

Abstract

We developed a language called the Automation Language for Managing Operations (ALMO). It was designed to automate the operations of communications links in NASA's Deep Space Network (DSN). ALMO was developed in response to a number of deficiencies that were identified with the previous languages and techniques used to manage DSN link operations. These included a need to: (1) provide visibility into the different link devices in order to recognize an anomaly and alert the operator when it occurs, (2) provide an intuitive and simple language capable of representing the full spectrum of operations procedures, (3) mitigate the variations in operating procedures experienced between different complexes and tracking supports, and (4) automate the overall operation, reducing cost by minimizing man hours required to configure devices and perform activities. With ALMO, for the first time DSN operators are able to capture sequences of activities into simple instructions that can be easily interpreted by both human and machine. Additionally, device information, which used to be available only via screen displays, is now accessible for operator use in automating their tasks, thus reducing the time it takes to perform such tasks while minimizing the chance of error. ALMO is currently being tested at the Deep Space Communications Complexes in Canberra, Australia and Goldstone, California.

Key Words: Automation, Deep Space Network, Subsystems, Link Operators

1. Introduction

[1] The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The Deep Space Network (DSN) is a worldwide network of spacecraft tracking and communications complexes located in Madrid, Spain; Canberra, Australia; and Goldstone, California. Each complex is capable of performing multiple missions simultaneously, all of which involve operating communications links. A DSN communications link is a collection of devices used to track and communicate with a spacecraft (Ref. 2). Examples of devices include antennas of various sizes, from diameters of 26 meters to 70 meters, transmitters and receivers. Such devices, per link, must share a complex's capabilities and time allotments/constraints.

The goal of ALMO is to automate DSN link operations by reducing the amount of time it takes to configure, calibrate, and operate the communications links used to support deep space communications missions (Ref. 1). Currently, operators perform hundreds of keyboard entries to configure the different DSN devices required to perform specific pre- or post-track activities. These keyboard entries are either "directives" (control messages to subsystems instructing them to perform specific tasks) or are commands to invoke displays that monitor the health and status of one or more devices. Current manual operations require that operators have a considerable knowledge of how to operate the different devices and their dependencies. Operators must also be aware of the many diagnostic messages devices send to the link controller in monitoring the link, and assure that directives sent to the devices have the intended effect. This process is error-prone, time-consuming, and labor-intensive.

Previous attempts were made to improve this process and to decrease the work load placed on link operators. The first language attempting to automate DSN operations was the Macro language. The Macro language provided limited flow control, imposed a limit on the size of a script, and provided only a few constructs for performing link activities. In addition, it did not include mechanisms for automatic parameterization. These limitations forced operators to create different versions of a script for use with different tracking supports, making maintenance of these scripts difficult.

The second language used was an off-the-shelf scripting language called BasicScript. At first, BasicScript seemed to meet the needs of automating link operations. But it lacked the constructs that automation needs to easily perform DSN operations, imposed size restrictions on scripts, limited usage of variables and subroutines, and required that operators understand every detail and all the code used to monitor and control the various devices. BasicScript also employed a one-layered architecture, which meant that a script contained all the encoded knowledge required to perform a task, and contained the code by which the script communicated to the transport layer underneath it. Additionally, all the monitoring and diagnostic functions were visible in the script, including invocation of graphical user interfaces used to obtain information from the operators. This made it tedious and difficult for operators to understand and modify a script, especially should last minute changes be required.

2. Goals

The challenge was to reduce operations cost by simplifying and minimizing the amount of time required to operate the DSN. We needed a tool that would reduce the operator workload, number of errors, and recovery time oftentimes associated with complex, manual operations and resource limitations. With these in mind we developed ALMO, which has the following features:

Extensible:	Capable of representing the full spectrum of operations procedures
Flexible:	Allows for the variations in operations procedures between different operations complexes and tracking supports
Robust:	Provides the constructs necessary to identify problems, accesses device monitor data and diagnostic information, tailors general procedures to specific circumstances
Maintainable:	Easy to update and maintain
User Natural:	Readable and usable by both computer and human operator (Ref. 3)

3. ALMO

3.1 Definition

ALMO stands for Automation Language for Managing Operations. It was implemented using the C language and an embeddable scripting language called Tcl/Tk (Tool Command Language). Tcl was chosen as the base scripting language because it: (1) allows new commands to be added with ease, (2) allows us to achieve an easier-to-understand syntax,[1] and (3) is a reliable and commercially-accepted product.

A disadvantage of adopting a commercially-available product is that bugs from other vendors may be incorporated into the software. (Fortunately, this is not a problem with which we have had to contend.) The advantages are: (1) reduction in time and cost of development, and (2) reduction in the testing and maintenance of the language.

A prototype was developed and successfully demonstrated. It was later ported into an operational version which is currently being tested.

---

[1] Link operators with little or no programming experience will be writing scripts, so easily understood commands and scripting language syntax are important.

3.2 Key Components

3.2.1 Network Monitor and Control (NMC), TDNs, Blocks

ALMO uses the NMC Software as its means of communicating to the various DSN link devices, or subsystems. It provides real-time monitor and control of the DSN. The NMC uses Temporal Dependency Networks (TDNs) to automate pre- and post-track calibration of subsystems. A TDN is a directed graph of interconnected nodes that represents an end-to-end sequence of operations. It also specifies sequential and parallel operations. Each node in a TDN is called a block. A block contains pre- and post-track conditions, and directives that are sent to the subsystems sequentially. As mentioned earlier, a directive is a control message sent to an individual subsystem instructing it to perform a specific task. A block may contain one or more directives to more than one subsystem and is written in ALMO.

3.2.2 Directive Responses, Monitor Data, Event Notice Messages

ALMO allows operators to access subsystem information that previously was only viewable from screen displays. This information can be used to monitor and control the subsystems, determine the logical flow of a block, and detect anomalies when they occur.

When an operator sends a directive to a subsystem, the subsystem returns a *directive response*, which is simply an acknowledgment that the directive was received. It does not indicate the successful or unsuccessful execution of a directive (Ref. 4). However, the need to associate a directive response to its specific directive is crucial in determining whether a directive was successfully sent or not. ALMO provides mechanisms that identify directive responses with their associated directives. Furthermore, it is important to determine whether a directive had the intended effect. ALMO uses either or both *monitor data* or *event notice messages* to verify successful or unsuccessful execution of directives. Monitor data is data provided by a subsystem that reflects its state and its parameter values and is stored in a pre-defined format. Event notice messages are textual messages that relay information about the state of a subsystem or a value of a subsystem parameter. However, when monitor data and event notice messages are generated, they are not explicitly tied to any directive. The operator must rely on his/her experience to determine which directive was most likely to have caused the subsystem to send the event notice message or generate the monitor data change. ALMO allows an operator to verify directive execution by specifying which monitor data items and event notice messages to check, and automatically alerts the operator if it detects an anomaly.

3.3 Architecture

ALMO commands are parsed, interpreted, and implemented via an engine. The engine is a UNIX process that runs on a workstation, in our case a SUN Ultra 2. The engine can be broken down into four (4) key areas: the command interpreter, event handler, intercommunication protocols, and the GUI.

3.3.1 Command Interpreter

The command interpreter is the component that includes the ALMO command library, and the interpreter that translates the ALMO command into machine language and executes it before going to the next ALMO command. The command interpreter incorporates Tcl, an embeddable and interpreted scripting language which provides generic programming facilities as variable substitution and control flow. An extension to Tcl called Tk is included to provide graphical routines. Lastly, commands that we designed and developed mainly for interacting with subsystems are added to the mix. The whole of these parts forms ALMO. Thus, the ALMO library comprises of the Tcl Interpreter, Tcl commands, Tk commands, and ALMO commands. Shown in Figure 1 is the ALMO Library.

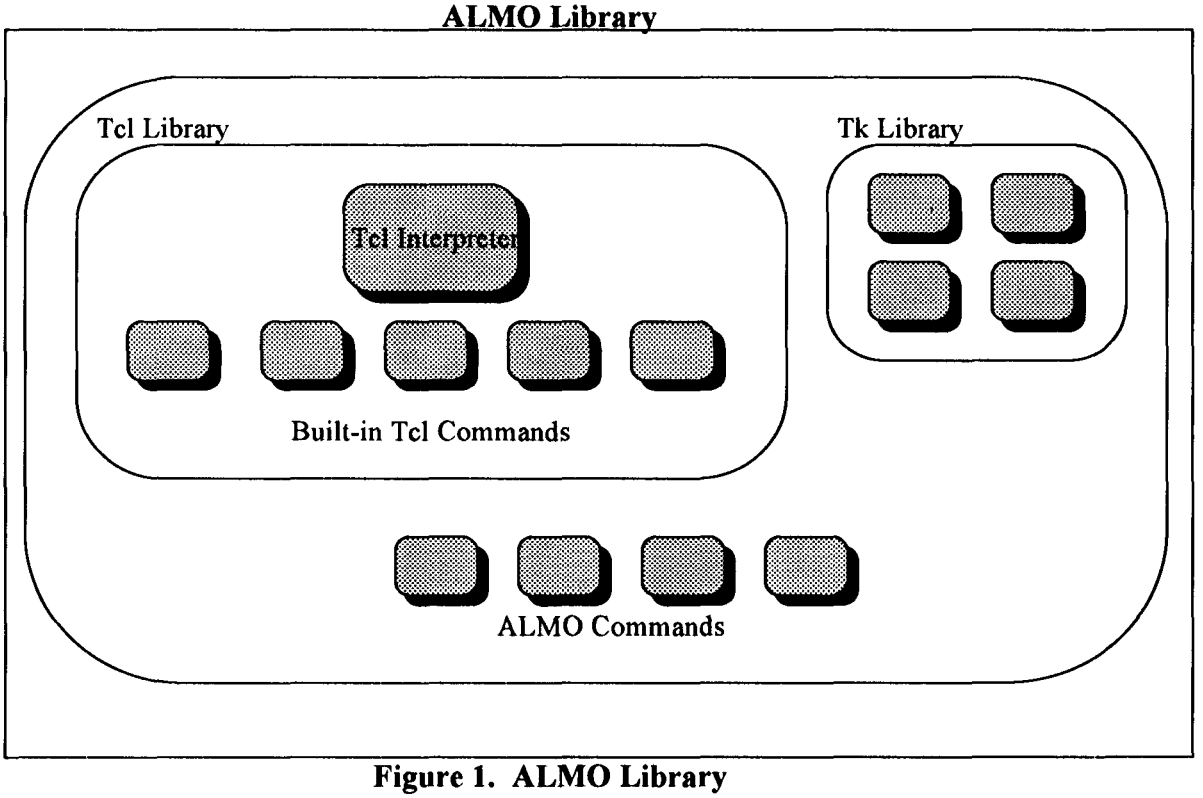

Figure 1. ALMO Library

3.3.2 Event handler

The event handler is the component responsible for collecting and processing "events" or notices of activity from subsystems, the graphical user interface (GUI), and/or the external and internal infrastructure. More importantly, the event handler is key in responding to events, allowing you to write event-driven blocks, i.e., instead of processing instructions sequentially, the event handler waits for events to come in and then acts upon them.

There are various forms of events. Internally there are events produced by the GUI, or there are messages from another process with information or a request. In addition, the engine should be able to respond to control commands from the user. Such commands include pausing and running. The script may also need to make parameter or display requests, for example, to another engine. Likewise, the engine should be able to receive and process the responses from other engines. Moreover, asynchronous communication is established, i.e., non-blocking communication, so any messages will be stored and kept in an I/O buffer until read. During script execution, the event-loop becomes active via calls to event-handling routines, otherwise events do not get processed until the command finishes. All of the above help in providing an event-driven engine that can process ALMO commands in a script, while at the same time react to any GUI requests or events from subsystems or other engines. We are indebted to the Tcl library for providing functions needed to accomplish a goodly portion of this.

In the NMC environment there are two engines that communicate with the ALMO engine: a connection engine, which is part of the NMC workstation software, and a TDN engine, which is part of the NMC Automation software. The connection engine is used to route and log directives to subsystems as well as collect event information for its operations logs. It is merely a method for the operator to collect directives from simultaneously running blocks and confirm their sources and destinations for security reasons. Otherwise, the ALMO engine is fully capable of routing the directives directly to the subsystems themselves. On the other hand, the TDN engine is used for routing user information collected from the operator, information tables, or monitor data from other outside sources. Figure 2 illustrates the ALMO architecture.

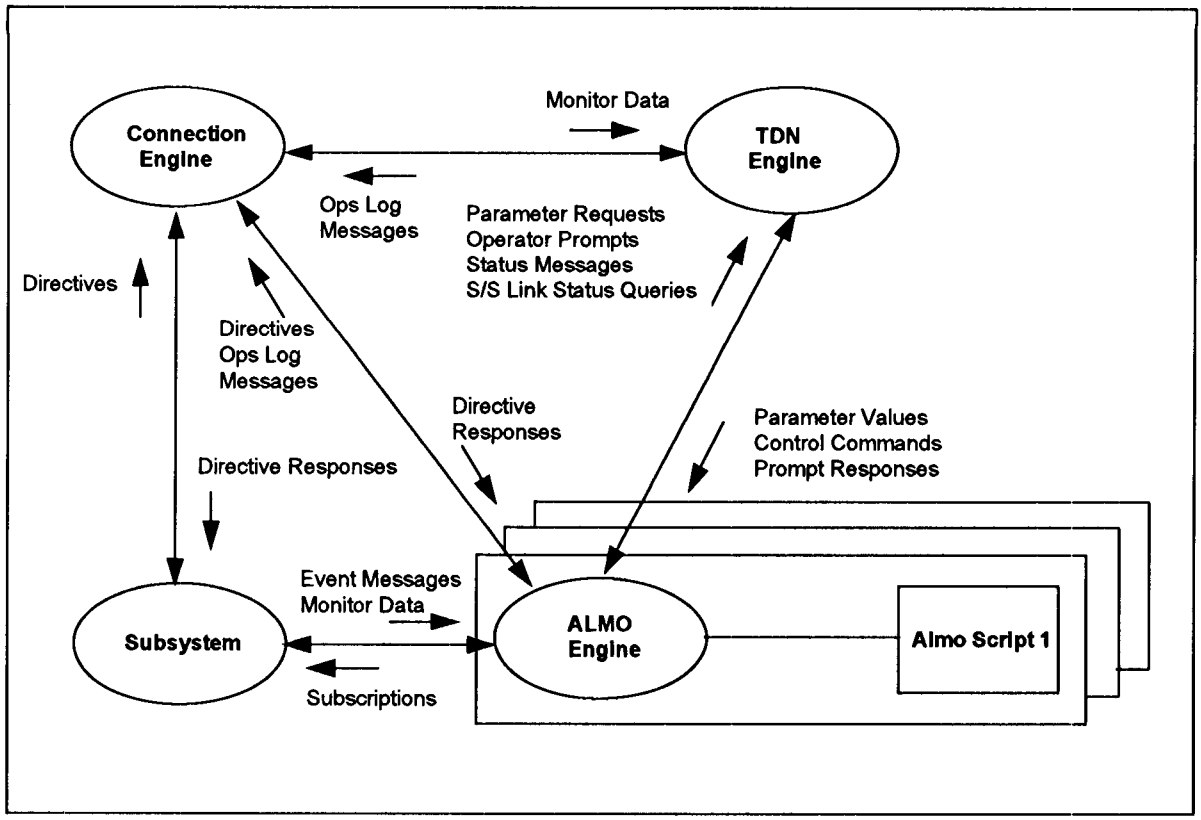

Figure 2. ALMO Architecture

3.3.3 Intercommunication Protocols

The ALMO engine can interact with a number of different processes and devices/subsystems. In order for it to exchange information with these, however, all participants must be able to understand each other. This is accomplished using intercommunication protocols that define a common language.

UNIX pipes are used to establish connection between the ALMO engine and the TDN engine, which is also a UNIX process, and text messages are used as the communication protocol. Since TDN is part of NMC Automation, and the NMC workstation employs the DSN Common Services for communicating with subsystems, the ALMO engine inherits this additional foundation, which includes the Monitor and Control Services (MCS). MCS is based on an implementation of the Distributed Computing Environment (DCE) and Distributed File System (DFS), and uses Remote Procedure Calls (RPCs) for interprocess communication. Figure 3 shows the intercommunication protocols used by ALMO.

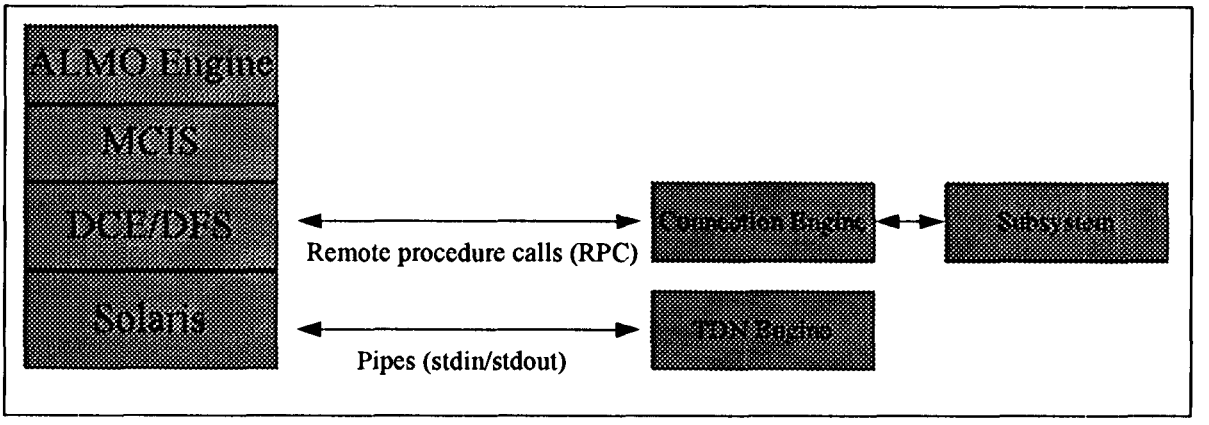

Figure 3. Intercommunication Protocols in ALMO

Outside applications are possible, but the intercommunication protocols and transport layer, defined for ALMO by Monitor and Control Infrastructure Services (MCIS) and DCE/DFS (depicted above), would have to be adapted through use of an Application Programming Interface (API).

3.3.4 Graphical User Interface

A graphical user interface is provided to display ALMO blocks that has the ability to show an active script during execution. A "bouncing ball" indicates which command is currently being processed. Perhaps most importantly, the GUI allows the user to control execution of the block, e.g. pause, resume, reduce or increase execution speed. The graphical user interface is developed entirely in Tk, the graphical extension to ALMO's Tcl code. A snapshot of the Block Viewer is shown in Figure 4.

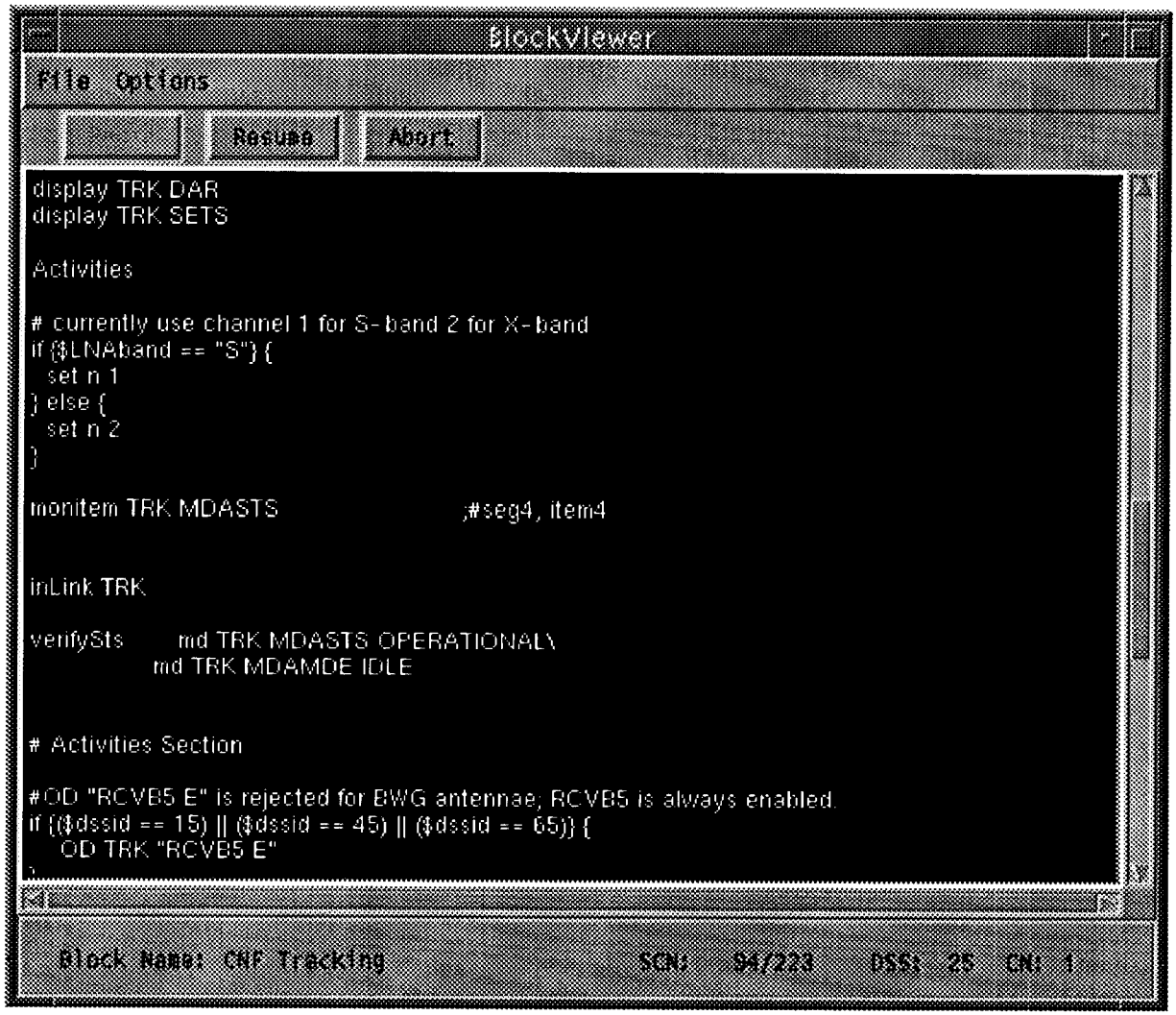

Figure 4. The Block Viewer

3.4 Language Description

3.4.1 Language Features

ALMO provides many constructs for performing DSN link operations. Some of the features are:

- specifying a block's associated displays information
- sending messages to the display log
- declaring, polling, subscribing, and fetching monitor data
- obtaining parameter inputs form either the operator or electronic source
- querying for a subsystem's link status, i.e., is the subsystem in the link or not?
- suspending and resuming block execution
- querying for a subsystem's short and long Directive Destination Code (DDC) name

- verifying directive status via monitor data, event message, and/or operator confirmation
- prompting the operator to make a selection among several options
- prompting the operator to enter a parameter value
- obtaining the current time
- comparing, adding, and subtracting time
- subscribing and unsubscribing to event notice messages
- calling one or more blocks within a block
- delaying block execution

Shown in Table 1 is a complete list of ALMO commands.

ALMO Command	Description/Syntax
startBlock	A required command that defines the start of a block
display	Defines and declares a display associated to a block
monitem	Declares, subscribes, polls, and fetches monitor data to be used in the block
param	Sends a parameter request
Activities	A required command that indicates the start of activities in a block
inLink	Queries the connection for Subsystem link status
getShortDDC	Returns: short DDC Name if subsystem is in the link
getLongDDC	Returns: long DDC Name if subsystem is in the link
selectOp	Invokes a dialog box and prompts the operator to make a selection
inputOp	Invokes a dialog box and prompts the operator to enter a string input
getUTC	Returns the current time in Universal Time Coordinated (UTC).
cmpUTC	Compares current time with a given time
addTime	Returns the new time after adding <period_of_time> to <time>.
subTime	Returns the new time after subtracting <period_of_time> from <time>
compare	Performs either string or numerical comparison
OD	Sends a directive
opsLog	Sends a message to the log
verifySts	Verifies the status of a directive or a subsystem using one or more monitor data, event notice message, operator confirmation
call	Runs another ALMO block (child block) while the caller block (parent block) waits for the child block to finish. When the child block is finished, the parent resumes execution.
release	Runs another ALMO block (child block) while the caller block (parent block) continues execution after it has "released" the child block. The parent block does not wait for the child block to finish and runs in parallel with the "released" block.
eventSubscribe	Subscribes to events from the event message server
eventUnsubscribe	Unsubscribes to an event
delay	Set delay in block execution
wait	Suspends block execution for a given period of time
endBlock	A required command that defines the end of a block

Table 1. ALMO Commands

3.4.2 Key Features

ALMO has several features that make it an ideal language for automating DSN operations. One is its ability to automatically parameterize blocks. In an effort to minimize human intervention, ALMO has provided constructs that allow an operator to specify electronic sources for parameters that are required in a block. The operator is also given the option to provide a default value in the event that the ALMO engine is unable to automatically obtain one from the specified source. If an operator runs more than one block, each of which requiring at least one parameter, these values are cohesively obtained and presented. The operator must intervene only once, to either confirm or modify the values.

ALMO also allows operators to use traced variables when appropriate. A traced variable registers a command to be called whenever a variable is accessed, modified, or unset (Ref. 5). Traced variables are especially handy when using monitor data in blocks. By declaring a monitor data item once in a block, the ALMO engine registers a subroutine[2] that is automatically invoked when a specific monitor data is used in a block.

ALMO also provides several mechanisms for accessing subsystem information that may be used for monitor, control, and diagnostic purposes. Information such as directive responses, monitor data, and event notice messages, that used to be available only by invoking subsystem displays, is now available to an operator. Access to this information provides the operator with a lot of flexibility and control. Additionally, it allows an operator to incorporate an error detection and operator notification capability in a block. However, ALMO already has built-in error detection capabilities within all of its commands. Whenever an anomaly is detected, the operator is alerted and presented with several possible courses of action.

In addition, the features that make ALMO an ideal language for automation also make it an ideal language for testing. ALMO allows a tester to invoke test functions, check results, and report errors. Test scripts can be developed rapidly and saved for regression testing. Used in whatever capacity, ALMO serves as a useful tool by applying a flexible and simple interface for interacting and manipulating hardware or software. For example, ALMO can be used to test new DSN devices or devices that have been upgraded before they can be used operationally in the DSN.

---

[2] The subroutine registered by the ALMO engine has the same name as the traced monitor data. For example, if a monitor data is declared as "MDASTS" in a block, a subroutine called "MDASTS" will be registered. A reference to the variable "MDASTS" ("$MDASTS", which means "the value of the monitor data named 'MDASTS'") will invoke a subroutine called "MDASTS" which will fetch the value of the monitor data from the subsystem.

3.4.3 Sample Block

A block written in ALMO is concise, readable, and easy for operators to understand, while ALMO's simple, high-level commands do most of the work. ALMO is able to greatly reduce the amount of code an operator has to write and/or understand. In fact, a nine-page block written in BasicScript was reduced to one page when converted to ALMO. The block in Figure 1, written in ALMO, configures the DSN's Metric Data Assembly (MDA) subsystem and is currently being tested at the Deep Space Complexes in Canberra, Australia and Goldstone, California.

```
1  startBlock "CNF Tracking"
2
3  #Parameter Inputs
4
5  # For three way tracking mode, get uplink dssid and append it.
6  monitem NMC SOE_Way
7  if {$SOE_Way == 3} {
8     monitem NMC SOE_3WayDSS
9     set way ${SOE_Way}W${SOE_3WayDSS}
10 } else {
11    set way $SOE_Way
12 }
13
14 param dssid "DSS ID" conn SOE_Dss
15 param predSetID "Predict Set ID" conn FranzCode
16 param LNAband "Downlink Band" conn SOE_LNAband
17 param bot "Beginning of Track" conn SOE_Bot
18 param sam "Sample Rate" conn SOE_Doppsample
19 param way "Track Mode/Way" op $way
20
21
22 #Associated Displays
23 display TRK CNF
24 display TRK DAR
25 display TRK SETS
26
27 Activities
28
29 # Currently use channel 1 for S-band, 2 for X-band
30 if {$LNAband == "S"} {
31   set n 1
32 } else {
33   set n 2
34 }
35
36 inLink TRK
37
38 monitem TRK_MDASTS          ;#seg4, item4
```

```
39
40 if ![compare OPERATIONAL $MDASTS] {
41   selectOp "The MDA is not operational" Continue Restart Abort
42 }
43
44 verifySts md TRK MDAMDE IDLE
45
46
47 # Activities Section
48
49
50 if {($dssid == 15) || ($dssid == 45) || ($dssid == 65)} {
51    OD TRK "RCVB5 E"
52 }
53
54 OD TRK "RANGE D"
55
56 OD TRK "SAM $sam" C:WAIT 7
57       verifySts md TRK [format "C%sSMP" $n] $sam
58
59 OD TRK "WAY $way" C:WAIT 7
60       verifySts md TRK [format "C%sWAY" $n] [string index $way 0]
61
62 OD TRK "PRED $predSetID" C:WAIT 7
63       verifySts md TRK DPRDID $predSetID
64
65 set time [subTime $bot 000:00:05:00]
66 #Same as variable "time" but with no colons and no DOY (per timed direct
format)
67 set formattedTime [subTime $bot 000:00:05:00 -noDOY -nocolon]
68
69 if {[cmpUTC $time] == 1} {
70       OD TRK "RUN"
71       wait 7
72            verifySts md TRK MDAMDE RUN
73 } else {
74       OD TRK "RUN NORPT"
75       wait 7
76            verifySts md TRK MDAMDE RNHR
77
78       getLongDDC sd TRK
79       OD NMC "^$formattedTime $sd RUN"
80 }
81
82 endBlock "CNF Tracking"
```

Figure 5. Block written in ALMO

This example defines parameters, sends directives to a subsystem identified as TRK (the generic subsystem name for the MDA), and checks the subsystem after each directive is sent. The first command executed will be `startBlock` which takes one argument, the name of the block, which it uses for logging purposes. In addition, the command handles initialization of the engine including establishing the communication route from the engine to the subsystem. On the other hand,

endBlock (line 82) is used to indicate the end of the block and handles shutdown of the engine as well as log the end of block execution. If the engine were to quit before endBlock is reached, the engine will exit the same way with the exception that it is considered an "abnormal exit".

The group of commands between startBlock (line 1) and Activities (line 27) is considered the declarations section, where parameter requests are collected and information is passed between the TDN engine and the ALMO engine. These param commands are compiled and processed when Activities (line 27) is reached. After the user has filled in values for the parameters, block execution then proceeds to the if statement on line 30 in what is considered the activities section. The if statement simply determines the value of n depending on what the variable LNAband is set to, as determined by the value of monitor data item SOE_LNAband (line 16).

To check the health of a subsystem, a monitor data item must be subscribed to and the monitor data value checked. This can be accomplished with the monitem command on line 38. The following if statement (line 40) compares the value of MDASTS to the text "OPERATIONAL". If it matches, then execution proceeds to the next command. Note that the same logic can be accomplished using a simpler implementation, "verifySts md TRK MDASTS OPERATIONAL".

If the subsystem is healthy, then the block will send a number of directives via the OD command. The directives configure the MDA for spacecraft tracking, but more importantly, after most of the directives is a verifySts command that checks the subsystem to confirm that the directive actually performed its intended action.

The commands starting at line 65 determine what directive to send based on what time it is. If the current time is not within 5 minutes of its designated start time (stored in the variable bot), then it will timestack a directive (line 79). Otherwise, it sends a "RUN" directive immediately. Lastly, as described above, if the block has executed the way it was intended, then endBlock (line 82) will be executed, and the block has finished running.

4. Status

Enhancements are being made even as ALMO is in the Pre-Acceptance Test phase. New commands, added to the already existing set, would enable operators to write event-driven blocks based on monitor data. In conclusion, some research tasks have expressed interest in using ALMO to develop blocks for fault, detection and recovery systems. All of this indicates that a ceiling has not yet been reached as to ALMO's flexibility and capability.

5. References

1. Cooper, Lynne P., Rajiv Desai and Elmain Martinez, "Operator Assistant to Support Deep Space Network Link Monitor and Control," SOAR Symposium, Houston, TX 1991.
2. Hill, Randall W., Jr., Kristina Fayyad, Patricia F. Santos and Kathryn Sturdevant, "Knowledge Acquisition and Reactive Planning for the Deep Space Network," Working Notes of the 1994 Fall Symposium on Planning and Learning: On to Real Applications, AAAI Press: New Orleans, LA 1994.
3. Fayyad, Kristina E., and Lynne P. Cooper, "Representing Operations Procedures Using Temporal Dependency Networks", SpaceOPS 92: Second International Symposium on Ground Data Systems for Space Mission Operations, Pasadena, CA, 1992.
4. Hill, Randall W., Jr., Steve Chien, Kristina Fayyad, Patricia F. Santos, and Crista Smyth, "Planning for Deep Space Network Operations", Proceedings of the 1995 AAAI Spring Symposium on Integral Planning Applications, Palo Alto, CA, March,1995,pp.51-56.
5. Welch, Brent B., "Practical Programming in Tcl and Tk", Prentice Hall, Inc.,: New Jersey, 1995, p. 79.